On the Complexity of the Natural Deduction Proof Search Algorithm

Bolotov, A., Shangin, V. and Kozhemiachenko, D.

This is an electronic version of a paper presented at *ARW2017 - 24th Automated Reasoning Workshop,* Bristol 03 to 04 Apr 2017, University of Bristol Technical Report. .

On the Complexity of the Natural Deduction Proof Search Algorithm

D.A. Kozhemiachenko,[1] A.E. Bolotov,[2] V.O. Shangin,[3]

[1] Lomonosov Moscow State University, kodaniil@yandex.ru
[2] University of Westminster, A.Bolotov@wmin.ac.uk
[3] Lomonosov Moscow State University, b.shngn@gmail.com

Abstract: We present our first account of the complexity of natural deduction proof search algorithms. Though we target the complexity for natural deduction for temporal logic, here we only tackle classical case, comparing the classical part of the proof search for temporal logic with the classical analytical tableau.

Rules of Natural Deduction System.

We commence with the review of the classical part of the natural deduction system for temporal logic [3] define below the sets of elimination and introduction rules, and where prefixes '*el*' and '*in*' abbreviate an elimination and an introduction rule, respectively.

Elimination Rules:

$$\wedge el_1 \frac{A \wedge B}{A} \qquad \wedge el_2 \frac{A \wedge B}{B} \qquad \neg el \quad \frac{\neg\neg A}{A}$$

$$\Rightarrow el \quad \frac{A \Rightarrow B, \quad A}{B} \qquad \vee el \quad \frac{A \vee B, \quad \neg A}{B}$$

Introduction Rules:

$$\vee in_1 \frac{A}{A \vee B} \quad \vee in_2 \frac{B}{A \vee B} \qquad \wedge in \frac{A, \quad B}{A \wedge B}$$

$$\Rightarrow in \quad \frac{[C], \quad B}{C \Rightarrow B} \qquad \neg in \quad \frac{[C], \quad B, \quad \neg B}{\neg C}$$

In '$\Rightarrow in$' and '$\neg in$' formula $[C]$ must be the most recent non discarded assumption occurring in the proof. When we apply one of these rules on step n and discard an assumption on step m, we also discard all formulae from m to $n-1$.

Searching Procedures

Searching Procedures update lists of formulae in the proof, list of goals (list_proof, list_goals) or both of them. Let $\bot$ abbreviate a dedicated goal, contradiction, and 'G_{cur}' abbreviate the current goal.

Procedure (1) simplifies structures of formulae in list_proof by an applicable elimination rule. **Procedure (2)** is fired when the current goal is not reached. Here we distinguish two subroutines. Procedure (2.1) applies when the current goal is not reached. Analysing the structure of the current goal we update list_proof and list_goals, respectively, by new goals or new assumptions. Subroutines (2.1.1)–(2.1.9) guide this process. The rules below have structure $\Gamma \Vdash \alpha \longrightarrow \Gamma' \Vdash \alpha'$ indicating that the rule modifies some given inference task $\Gamma \Vdash \alpha$ to a new inference task $\longrightarrow \Gamma' \Vdash \alpha'$.

(2.1.1) $\Gamma \Vdash \Delta, A \longrightarrow \Gamma, \neg A \Vdash \Delta, A, \bot$

(2.1.2) $\Gamma \Vdash \Delta, \neg A \longrightarrow \Gamma, A \Vdash \Delta, \neg A, \bot$

(2.1.3) $\Gamma \Vdash \Delta, A \wedge B \longrightarrow \Gamma \Vdash \Delta, A \wedge B, B, A$

(2.1.4.1) $\Gamma \Vdash \Delta, A \vee B \longrightarrow \Gamma \Vdash \Delta, A \vee B, A$

(2.1.4.2) $\Gamma \Vdash \Delta, A \vee B \longrightarrow \Gamma \Vdash \Delta, A \vee B, B$

(2.1.5) $\Gamma \Vdash \Delta, A \Rightarrow B \longrightarrow \Gamma, A \Vdash \Delta, A \Rightarrow B, B$

If applying Procedure (2.1.4) we could not reach goals $A,\ B$ then we delete these goals, leaving the current goal, $A \vee B$.

Procedure 2.2 is invoked when $G_n = \bot$. It searches for formulae in list_proof as sources for new goals. We abbreviate these designated formulae as Ψ. The idea behind this procedure is to search for "missing" premises to apply a relevant elimination rule to Ψ.

(2.2.1) $\Gamma, \neg A \Vdash \Delta, \bot \longrightarrow \Gamma, \neg A \Vdash \Delta, \bot, A$

(2.2.2) $\Gamma, A \vee B \Vdash \Delta, \bot \longrightarrow \Gamma, A \vee B \Vdash \Delta, \bot, \neg A$

(2.2.3) $\Gamma, A \Rightarrow B \Vdash \Delta, \bot \longrightarrow \Gamma, A \Rightarrow B \Vdash \Delta, \bot, A$

Applying the Procedure (2.2.1) we have $\neg A$ in the proof and are aiming to derive, A itself. If we are successful then this would give us a contradiction.

When we apply Procedures (2.2.2-2.2.3), our target is to derive formulae that being in the proof would enable us to apply a relevant elimination rule, $\vee_{el}, \Rightarrow_{el}$.

Procedure 3 checks reachability of the current goal in list_goals. If $Reached(G_n) =$ **true** then list_goals= list_goals- G_n and $G_{cur} = G_{n-1}$.

Procedure 4 guides the application of introduction rules. Any application of the introduction rule is completely determined by the current goal in list_goals. This property of our proof searching technique protects us from inferring by introduction rules an infinite number of formulae in list_proof.

Proof-Searching Algorithm [3]

Given a task $\vdash G$, we commence the algorithm by setting the initial goal, $G_0 = G$. Then for any goal G_{cur}, we apply Procedure 3, to check if it is reached. If G_i is not reached we apply Procedure 1. If G_{cur} is still not reached, then Procedure 2 is invoked which updates list_proof and list_goals dependent on the structure of G_{cur}. If G_{cur} is reached, then Procedure 4 is applied. Otherwise, which could only be in the case, when *current_goal* is set as $\bot$ and we do not have contradictory formulae in list_proof, we update list_goals looking for possible sources of new goals in list_proof. Continuing searching we may reach the initial goal, G_0, in which case we terminate having found the desired proof. Otherwise, we reach the stage when our search cannot update list_proofand list_goals any further. In the latter case we terminate, and no proof has been found and a counterexample can be extracted.

Marking technique introduces and eliminates special marks for formulae in list_proof and list_goals. Most of

these marks are devoted to prevent looping either in application of elimination rules or in searching. In particular, we mark: formulae that were used as premisses of the rules invoked in Procedure 1; goals $A \vee B$ in Procedure (2.1.4); those formulae in list_proof which were considered as sources of new goals in Procedure 2.2 and these new goals themselves to prevent looping in Procedure (2.1.1).

Let 'last(list_goals)' return the last element of list_goals, and list_goals — G_n deletes the last formula, G_n, from list_goals.

Now, based on the procedures (1)-(4) we introduce the proof search algorithms $\mathsf{NPComp}_{\mathsf{ALG}}$.

(0) list_proof(), list_goals(), **GO TO** (1)

(1) Given a task $\Gamma \Vdash G_0$, $G_{cur} = G_0$ $(\Gamma \neq \emptyset) \longrightarrow$ (list_proof $= \Gamma$, list_goals $= G_0$, **GO TO** (2)) else list_goals $= G_0$, **GO TO** (2).

(2) Procedure (3): Reached (G_{cur}) = **true** $\longrightarrow$ list_goals = list_goals $- G_{cur}$

$(G_{cur} = G_0) \longrightarrow$ **GO TO** $(6a)$ else $G_{cur} = last$(list_goals) **GO TO** (3)

Reached (G_{cur}) = **false** $\longrightarrow$ **GO TO** (4).

(3) Procedure (4): apply an introduction rule, **GO TO** (2).

(4) Procedure (1): elimination rules

(4a) Elimination rule is applicable, **GO TO** (2) else **GO TO** (5).

(5) Procedure (2): update list_proof and list_goals based on the structure of G_{cur}

(5a) Procedure (2.1): analysis of the structure of G_{cur}, **GO TO** (2) else

(5b) Procedure (2.2): searching for the sources of new goals in list_proof), **GO TO** (2) else

(5c) (if all compound formulae in list_proof are marked, i.e. have been considered as sources for new goals), **GO TO** (6b).

(6) Terminate ($\mathsf{NPComp}_{\mathsf{ALG}}$).

(6a) The desired ND proof has been found. EXIT,

(6b) No ND proof has been found. EXIT.

Complexity Analysis

We consider a family Σ_n of formulas introduced by Cook and Reckhow in [4].

$$\Sigma_n = \bigcup\{\pm A \vee \pm A_{\pm} \vee \ldots \vee A_{\pm(n-1)\pm}\}$$

Here $+A = p$ and $-A = \neg p$ are literals. One can exemplify this family with $\Sigma_1 = \{A, \neg A\}$ and $\Sigma_2 = \{A \vee A_+, A \vee \neg A_+, \neg A \vee A_-, \neg A \vee \neg A_-\}$. Informally, Σ_n is simply a family of all disjunctions of literals with n disjuncts. It is clear that $|\Sigma_n| = 2^n$. We will further designate each member of Σ_n with F_i^n $(1 \leqslant i \leqslant 2^n)$. Following Cook and Reckhow, analytic tableaux can show inconsistency of Σ_n in at least $2^{\Omega(2^n)}$ steps.

We follow Massacci [5] (see also the discussion about Massacci's paper in [2]) and assume that literals are associated from left to right. Under these conditions Massacci showed that analytic tableaux can prove inconsistency of Σ_n in no more than $O(2^{n^2})$ steps which was exponentially shorter than lower bound provided by Cook and Reckhow.

We will further associate each Σ_n with two formulae:

$$\mathcal{F}_{\vee} = \bigvee_{i=1}^{2^n} F_i^n \quad \text{and} \quad \mathcal{F}_{\wedge} = \neg \bigwedge_{i=1}^{2^n} F_i^n$$

We assume that all F_i^n are associated and ordered arbitrarily in both cases.

It was shown that the proof searching algorithm for natural deduction is complete [1], i.e., that it can prove every classical propositional tautology. The algorithm has a remarkable property: it can delete steps of a derivation if it finds the current goal to be unreachable. This property means that there can be a difference between number of steps in the resulting inference and the number of formulas which were introduced to the inference.

The following theorems can be proved.

Theorem 1. *Proof searching algorithm can prove $\mathcal{F}_{\vee}$ in $O(2^n)$ steps including deleted ones.*

Theorem 2. *Proof searching algorithm can prove $\mathcal{F}_{\wedge}$ in $O(n \cdot 2^n)$ steps including deleted ones.*

References

[1] A.Bolotov, V. Bocharov, A. Gorchakov, and V. Shangin. Automated first order natural deduction. In *Proceedings of the 2nd Indian International Conference on Artificial Intelligence, Pune, India, December 20-22, 2005*, pages 1292–1311, 2005.

[2] N. Arai, T. Pitassi, and A. Urquhart. The complexity of analytic tableaux. *J. Symbolic Logic*, 71(3):777 – 790, 2006.

[3] A. Bolotov, O. Grigoriev, and V. Shangin. Automated natural deduction for propositional linear-time temporal logic. In *14th International Symposium on Temporal Representation and Reasoning (TIME 2007), 28-30 June 2007, Alicante, Spain*, pages 47–58, 2007.

[4] S.A. Cook and R. Reckhow. On the lengths of proofs in the propositional calculus. In *Proc. 6th ACM Symp.on Theory of Computing (STOC-74)*, pages 135–148, 1974.

[5] F. Massacci. The proof complexity of analytic and clausal tableaux. *Theoretical Computer Science*, 243(1):477 – 487, 2000.